MBASIC™ Batch Processor Architectural Overview

S. M. Reynolds
DSN Data Systems Section

The MBASIC™ Batch Processor will allow users to run MBASIC™ programs more cheaply. It will be provided as a CONVERT TO BATCH command, usable from the ready mode. It will translate the users program in stages through several levels of intermediate language and optimization. Only the final stage is machine dependent; therefore, only a small effort will be needed to provide a Batch Processor for a new machine.

I. Introduction

Many large, frequently used programs, which are slow and expensive to run, currently exist in the MBASIC™ user community. The MBASIC™ Batch Processor will reduce the time and cost associated with running these programs by providing a compiling facility within the MBASIC™ environment which will allow a user to convert programs to a directly executable form. Direct execution typically reduces program run times by at least an order of magnitude below those of the same programs executed interpretively.

Within the MBASIC™ environment, the user may invoke the Batch Processor through the following commands, prescribed in *Fundamentals of MBASIC™*, Vol. II (JPL, Feb. 19, 1974):

CONVERT ⟨filename⟩ TO BATCH ⟨filename⟩

CONVERT TO BATCH ⟨filename⟩

QUEUE BATCH ⟨filename⟩

RUN ⟨filename⟩

II. MBASIC™ Batch Processor Internal View

When called, the MBASIC™ Batch Compiler receives the pseudo-op code (POPCODE) buffer, the Line Reference Table (LRT), and the Name Reference Table (NRT) from the parser. The source code of the program to be compiled is never examined. The output of the compiler is directly executable code for the host machine.

Within the compiler, a completely machine-independent stage is followed by a machine-dependent stage. A major goal of the compiler design is to minimize the proportion of the machine-dependent stage.

The machine-independent stage uses the POPCODE, LRT, and NRT to generate a low-level, machine-independent assembly language called MCODE, which is then translated by the machine-dependent stage into host code (see Fig. 1).

The first step in the translation of POPCODE to MCODE is translation of POPCODE to an equivalent three-address code. This intermediate notation is easier to handle in the ensuing optimization step than the reverse polish notation of POPCODE because references to operands are explicit and are

directly associated with the operation code rather than implicit in pushes onto the stack by previous op codes as in POPCODE.

The second step in translation to MCODE is the optimization of the intermediate code. Because the input and output of this step are intermediate code, expressed in the same internal representation, this step will be delayed in the initial implementation and added later when the surrounding steps have been verified.

The third machine-independent step is translation of the intermediate code to MCODE.

Translation of MCODE to machine-dependent assembly language is performed in two steps: direct translation of MCODE to host code followed by machine-dependent optimization.

For example, the source statement

```
A=B+C
```

becomes

```
PUSH B
PUSH C
ADD
POP A
```

in POPCODE. This is translated into

```
ADD A,B,C
```

in the intermediate code, which is translated into

```
FLOAT TEMP1,B
ADD A,TEMP1,C
```

in MCODE (assuming that B is declared to be an integer), and into

```
LMJ X10,FLOAT
B
TEMP1
L A0,TEMP1
A A0,C
S A0,A
```

Before translation begins, a declaration processing pass through the POPCODE builds a Symbol Definition Table (SDT), which contains type and binding information for each variable explicitly or implicitly declared in the program. The translation process is performed one statement at a time in order to minimize the size of buffers and the use of external mass storage.

III. Pseudo-Op Code

MBASIC™ pseudo-op code is a reverse polish notation for the MBASIC™ virtual stack machine (VSM). One POPCODE string is produced for each MBASIC™ source statement. The MBASIC™ VSM checks for incompatible types at execution time and performs legal type conversions; it can handle arrays and strings as atomic data items; and it handles FOR-NEXT and GOSUB transfers implicitly by using auxiliary stacks. The reflection of these features in POPCODE makes it a relatively high-level language, requiring considerable translation to reach the level of a typical real machine. Even syntactically correct MBASIC™ programs may translate to POPCODE programs which generate runtime errors in the VSM. These errors will be detected and an error message printed during translation rather than at runtime of the compiled program, whenever possible.

IV. Intermediate Language

The Intermediate Language (IL) is a direct mapping of POPCODE to an equivalent non-stack-dependent language. All implicit references to stack data are replaced by explicit references to identifiers, either source defined or temporary identifiers defined by the IL processor.

With one exception, the translation from POPCODE to IL involves a one-to-one mapping of POPCODE statements to corresponding IL statements. The exception is that POPCODE operations whose only result is to push a variable's value onto the data stack do not appear in the IL string, but the identifiers are referenced as IL arguments.

V. POPCODE to IL Translation

The POPCODE to IL translation algorithm is a pseudo-execution of the POPCODE string using an identifier stack instead of a data stack. The popcode string is scanned, and each POPCODE encountered generates the corresponding IL. If values would have been popped from the data stack during execution, identifiers are popped from the identifier stack in the same order. If values would have been pushed onto the stack during execution, temporaries are created and their identifiers pushed onto the identifier stack.

VI. MCODE

MCODE is a low-level, three-address code for a class of virtual machines with variable storage attributes, such as:

(1) Differing number and type of registers.

(2) Differing mappings from main storage to registers (e.g., byte addressable machines which map several cells into one register and word addressable machines which map one cell into one register).

(3) Differing main storage size.

The MCODE generator for a specific implementation of the Batch Processor will be configured by setting its variable attributes to match the host system as closely as possible.

The MCODE virtual machine recognizes the following data types:

(1) Integer – at least 32 bits precision

(2) Real – host system floating point format

(3) Pointer – at least 16 bits precision

(4) Char – at least 7 bits precision

MCODE machines have a separate address space for each data type and for instruction code, and the unit quantity for pointer type and for any absolute address represents one cell of address space of the type pointed to.

Registers may be defined to accept one or more different data types, e.g., a general purpose register in many machines would be represented in the MCODE machine by a register accepting real, integer, and pointer data.

MCODE contains binary, unary, and nonary operations of the following kinds:

(1) Arithmetic
  (a) Add
  (b) Subtract
  (c) Multiply
  (d) Divide
  (e) Unary minus

(2) Boolean
  (a) And
  (b) Or
  (c) Exclusive or
  (d) Complement
  (e) Conditional jump
  (f) Unconditional jump

Each instruction is defined separately for each type, and all operands must be of matching type. Mixed type operations are possible through use of the following conversion operations:

(1) Integer to real

(2) Real to integer

(3) Integer to char

(4) Char to integer

(5) Pointer to integer

(6) Integer to pointer

Each instruction consists of an operation code and zero to three operands. Each operand contains a register/address flag, a register number or address, a type, and a boolean indirect addressing indicator, which may be true only if the operand specifies a pointer register. For arithmetic and boolean operations, the first operand specifies a destination for the result of the operation, and the two other operands specify the source locations. For unary operations, only one source operand is necessary. Jump instructions have one operand, which may only be a pointer register in the case of an indirect address reference or an address in the case of a direct address reference.

MCODE also contains the following operations on data files:

(1) Open

(2) Close

(3) Remove

(4) Rename

(5) Append

These operations have only one operand, which contains a file name.

The following operations have a file name operand followed by a source or destination operand:

(1) Get char

(2) Put char

VII. IL to MCODE Translation

IL to MCODE translation is an in-line expansion performed in a single scan of the IL sequence. Each IL is expanded into one or more MCODE statements. Code for type checking and type conversions for legal mixed type operations is generated during this process. Error messages for illegal types are

generated during type checking. The IL arguments generated are selected on the basis of the best known source for a value at compiled code execution time, i.e., a value known to be in both register and main storage will be represented by the register.

VIII. MCODE to Host Assembly Language Translation

In most cases, the host code will be its standard assembly language. MCODE to host code translation is a machine-dependent process; however, in most cases it can be performed in one scan of the MCODE string with one or more host code statements generated for each MCODE statement. This is possible because MCODE is generated for a virtual machine having storage characteristics similar to those of the real machine.

IX. Machine-Independent Optimization

The machine-independent optimization step is transparent (except in efficiency) to the rest of the MIDB processor because its input and output have the same representation. Optimization will concentrate on expressions and assignment statements. Many statistical studies of actual source code and the possible optimizations have been made, and all show these to be the most effective optimizations in terms of actual reduction of the size and execution time of typical object code. The selected optimizations are:

(1) Subexpressions appearing more than once are calculated once and saved in a temporary variable; instances of the common subexpression are replaced by a reference to the temporary variable.

(2) Subexpressions involving only constants are calculated at compile time.

(3) Multiple assignments to the same variable are recognized and replaced by the last assignment only.

X. Machine-Dependent Optimization

Machine-dependent optimization is also a transparent process (except for efficiency). Some possibilities common to most machines, but still machine-dependent, are the following:

(1) Chains of branches are replaced by a single branch instruction.

(2) Unreachable code is deleted.

(3) Common code groups preceding a collecting node are recognized, and the node is moved to precede the first instance of the common group. The other instance is deleted.

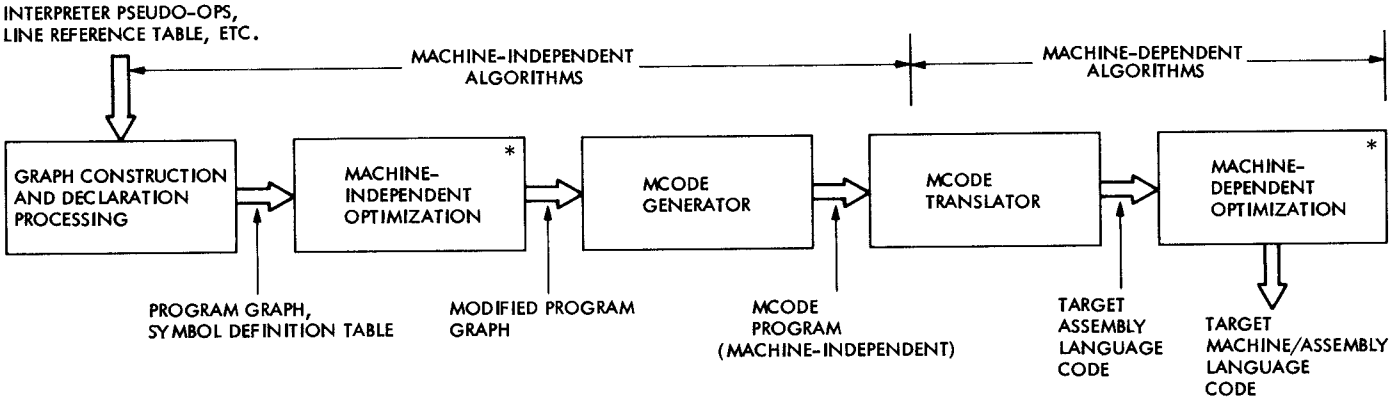
(4) Common code groups heading all branches of a program fork are condensed as in (3), above.

Other completely machine-dependent optimizations might involve such things as invoking auto increment mode in place of explicit pointer arithmetic or recognizing LOAD-STORE groups and replacing them with direct memory to memory moves.

XI. Summary

The MBASIC™ Batch Processor is a language translator designed to operate in the MBASIC™ environment. It will provide a facility with which MBASIC™ programs which have been debugged in the interpretive mode may be translated into a directly executable form and stored or executed from the MBASIC™ environment.

The Processor is to be designed and implemented in both machine-independent and machine-dependent sections. The architecture is planned so that optimization processes are transparent to the rest of the system (except for efficiency) and need not be included in the first design-implementation cycle.



* OPTIMIZATION OPTIONAL

MCODE CHARACTERISTICS:

MACHINE-INDEPENDENT LANGUAGE
LOW-LEVEL LANGUAGE WITH SIMPLE TRANSLATOR
TRANSLATOR IS A REQUIRED PART OF COMPILER
LANGUAGE IS THUS UNDER DSN CHANGE CONTROL
MODULES CODED IN MCODE ARE PORTABLE ONCE MCODE TRANSLATOR IS INSTALLED
REIMPLEMENTATION REQUIRES ONLY TRANSLATOR AND FINAL OPTIMIZER BE REIMPLEMENTED

Fig. 1. MBASIC™ Batch Compiler